SPETABARU: A Task-based Runtime System with Speculative Execution Capability

Bérenger Bramas

SPETABARU: A Task-based Runtime System with Speculative Execution Capability

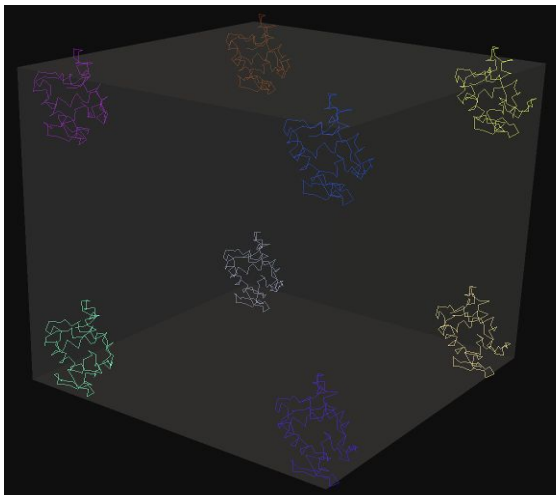
Bérenger Bramas
Inria Nancy - Grand Est / CAMUS Team

Summary

- Background & motivation
  - Monte Carlo (MC), Replica Exchange Monte Carlo (REMC)
  - Task-based parallelization (STF)
- Speculative execution in task-based runtime systems
  - Introduction
  - Challenges
  - Some solutions
- SPETABARU: SPEculative TAsk BAsed RUntime
  - Illustration
  - Performance study
- Conclusion & perspective

Motivation - Biophysic application

- Can be viewed as a molecular dynamic code
- A protein is composed of domains/polypeptides/chains of amino acids
  - Each domain is composed of beads/particles
  - A domain can “move/rotate/shift”
  - The global energy is computed with N^2 particles-particles interactions
- Too many configurations -> MC



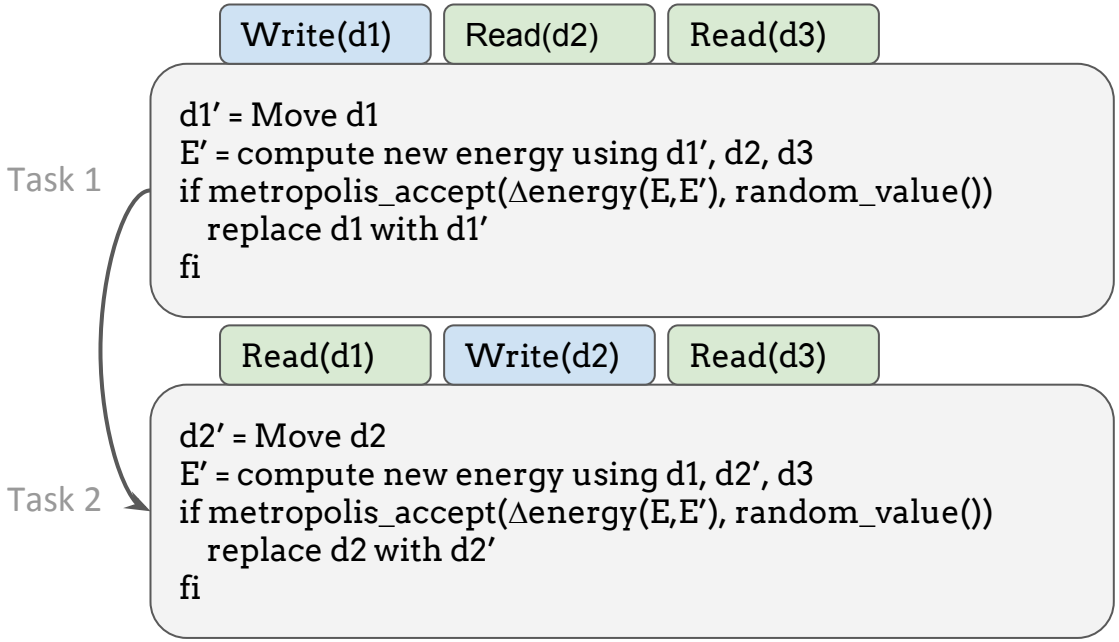
Motivation - MC

Write(d1) Read(d2) Read(d3)

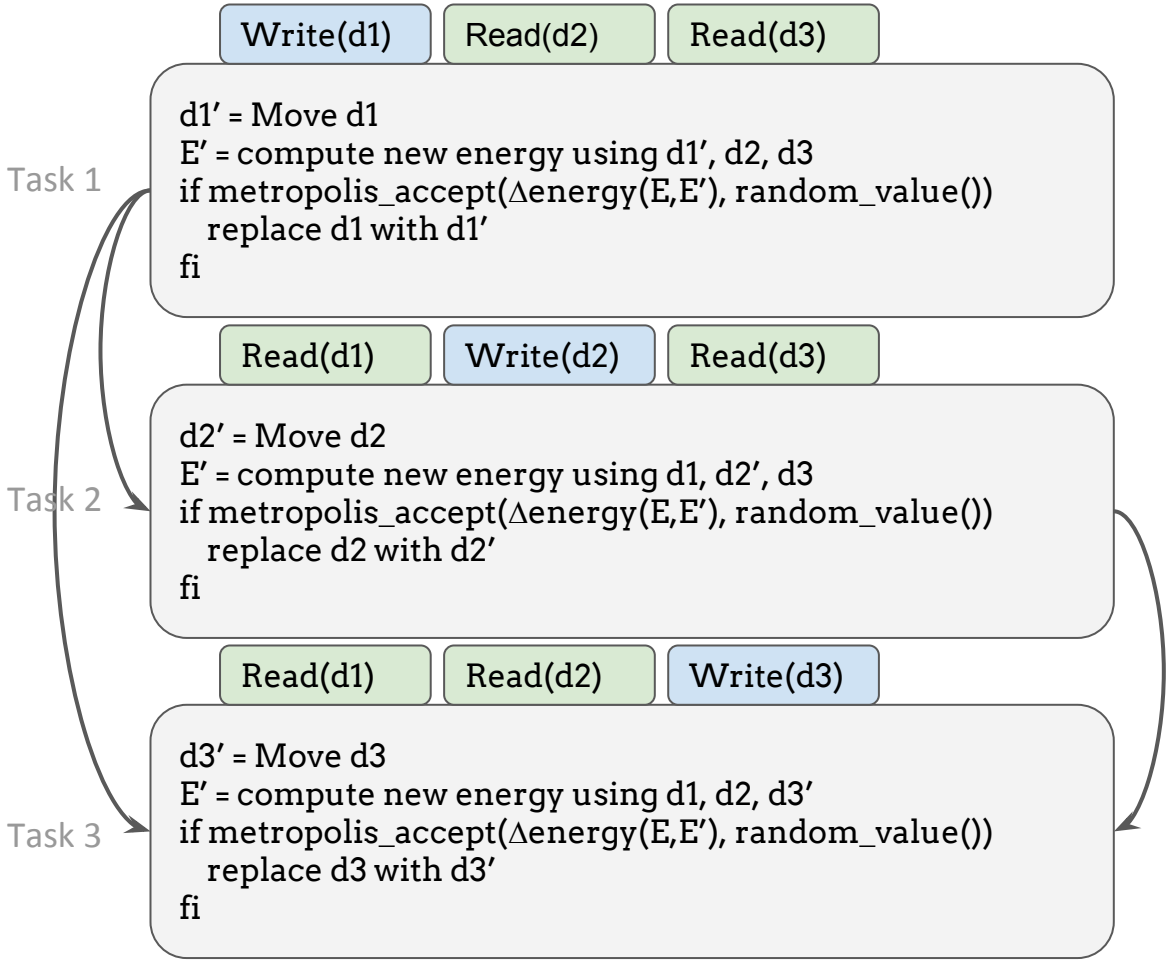
Task 1

```
d1' = Move d1
E' = compute new energy using d1', d2, d3
if metropolis_accept(Δenergy(E,E'), random_value())
   replace d1 with d1'
fi
```

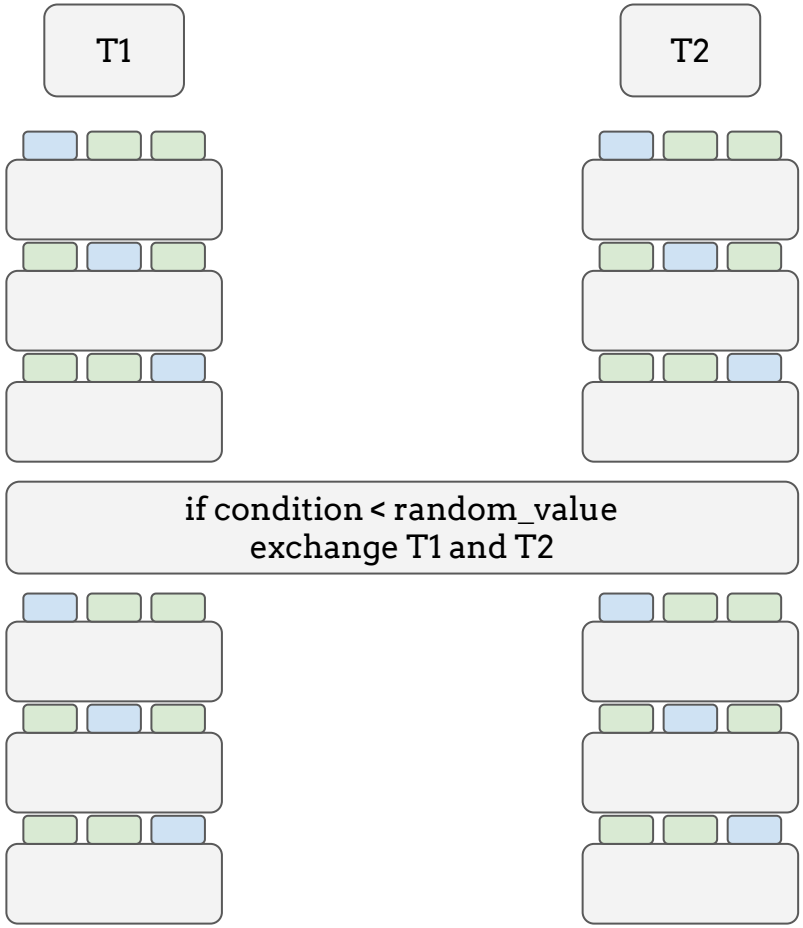
Motivation - MC



Motivation - MC



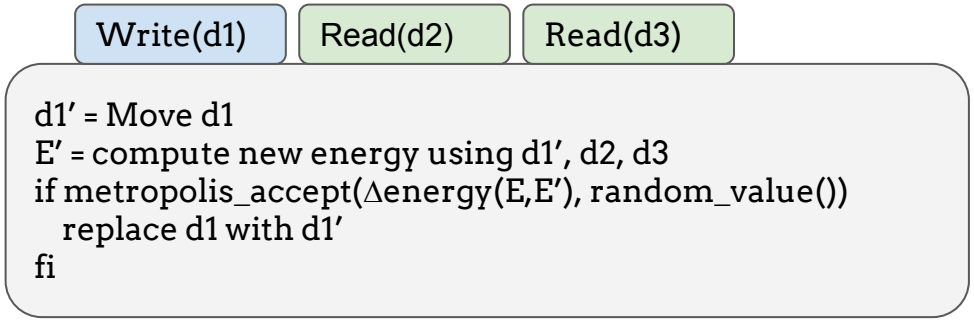
Motivation - REMC



Problem

- Limited degree of parallelism
  - Each iteration depends on the previous ones
  - Possible exchange of temperature = dependency

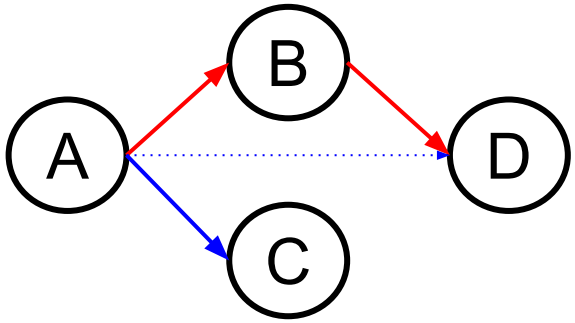
- Some tasks access the data in “write” but it is unknown at task insertion time if they are really going to modify the data or not



Background - Sequential task-flow (STF)

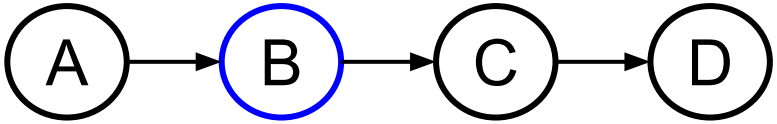
- The code is split into tasks
- Each task accesses the data in **READ** or **WRITE (~READ-WRITE)**
- **The tasks are created by a single thread** that informs the runtime system about the operations to do (which task to call with what data)
- The runtime system infers the dependencies between the tasks and ensures a correct execution (**sequential consistency**)
- DAG (node = task, edge = dependency)

```
add task A: write(m), read(n) {
   m += n;
}
add task B: write(n) {
   n++;
}
add task C : write(l), read(m){
   if(m > 0) l++;
}
add task D : read(m), read(n){
   print(m, n);
}
```



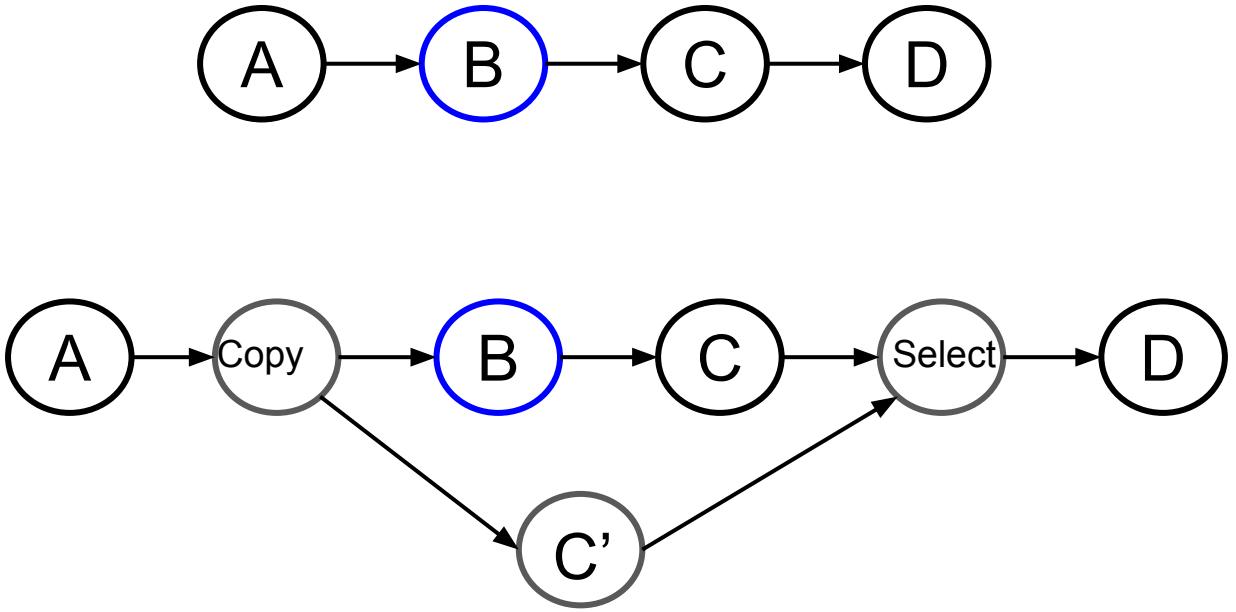
TB Speculative execution example

- Consider here that a unique data is used by all tasks
  - Task B is an uncertain task (it is not sure if it will write on the data)



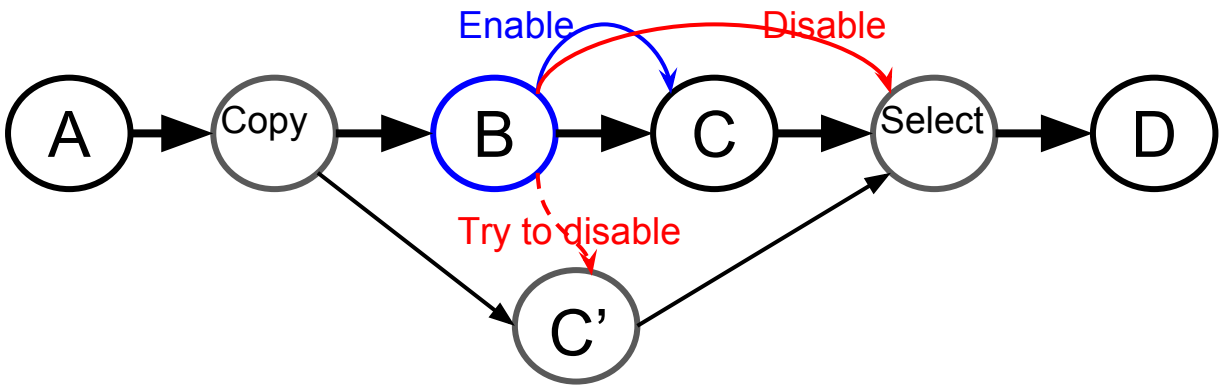
- Idea:
  - The programmer indicates that a task will potentially WRITE on a data
  - At the end of the task, the task informs the RS if the data was modified or not

TB Speculative execution example

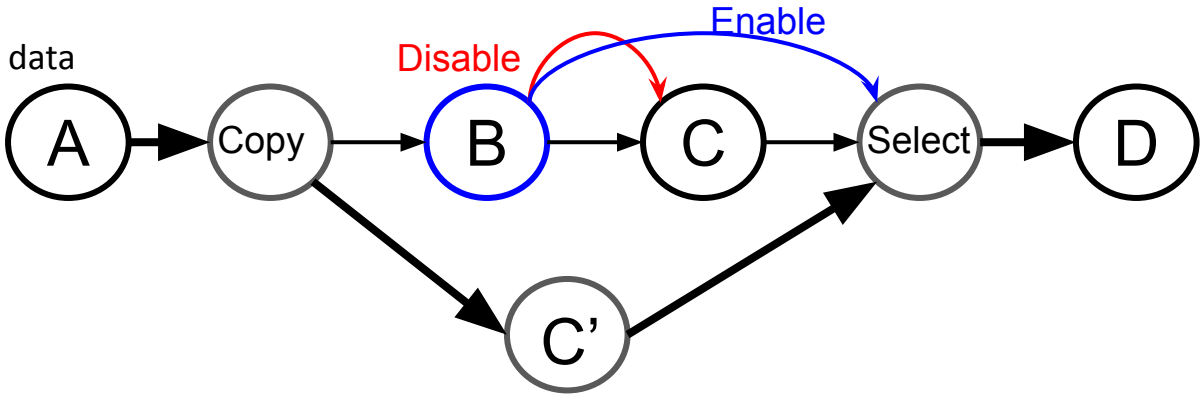


TB Speculative execution example

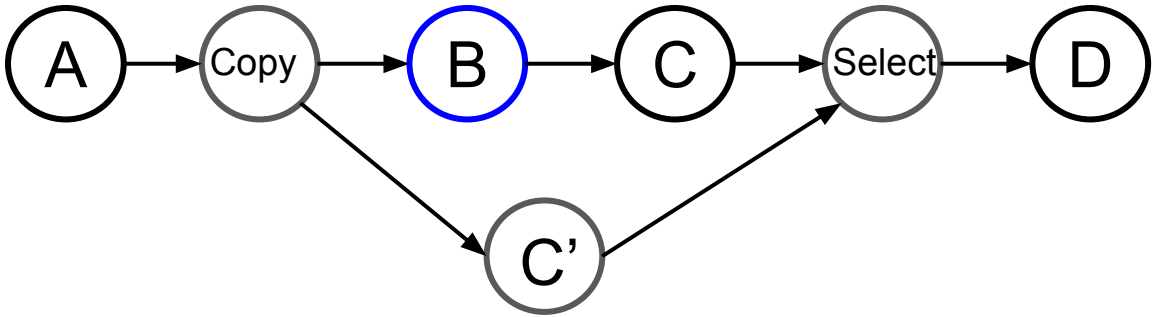
B wrote on the data



B did not write on the data



TB Speculative execution example



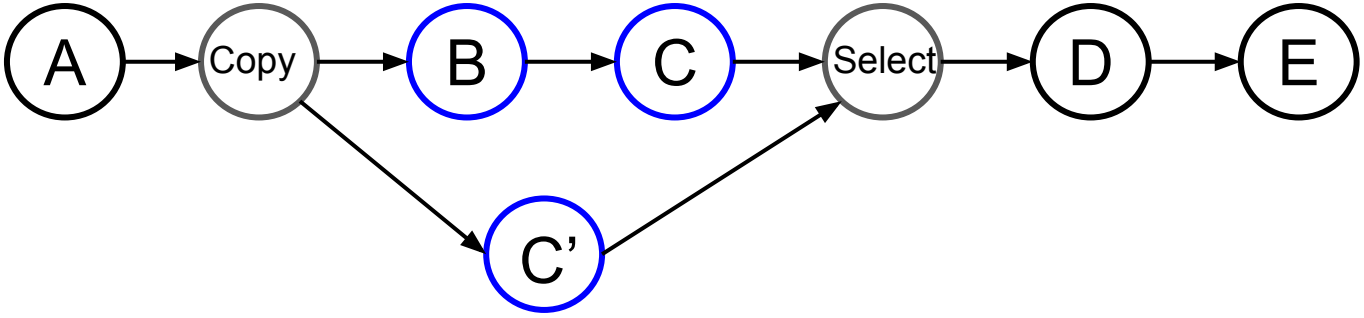
- Execution time:
  - Without speculation : T(B) + T(C)
  - With speculation :
    - If B writes on the data : T(copy) + Max(T(B) + T(C), T(C')) (T(C') is zero if canceled)
    - else : T(copy) + Max(T(B),T(C')) + T(select)

Challenges

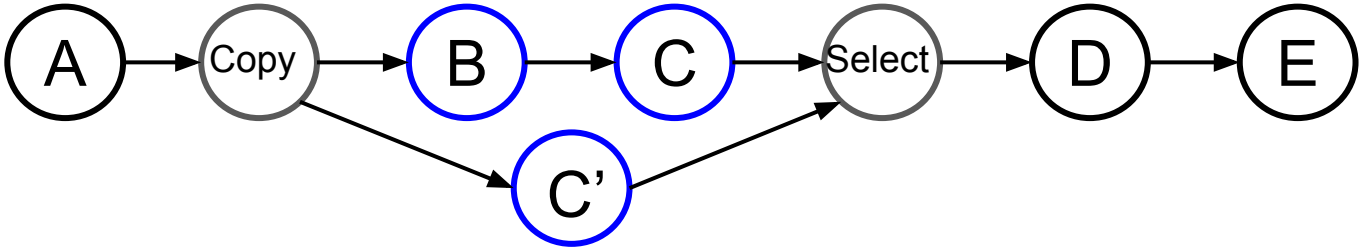
- We do not have the full DAG from the beginning
  - Tasks are inserted dynamically (and executed)
  - We have to manage all the dependencies
- What to do when there are more than one uncertain task ? (partially solved)
- Unsolved/WIP
  - Schedule a DAG with speculation
  - Predict success/failure if speculation

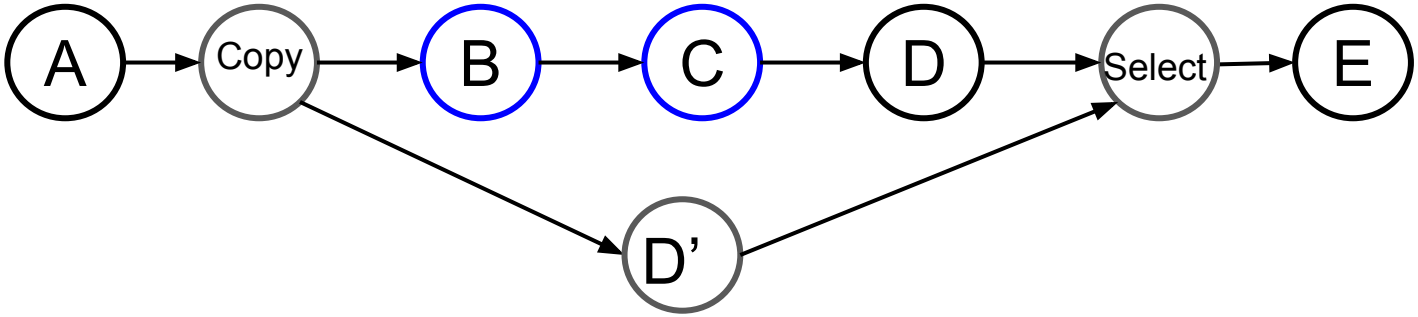
TB Speculative execution example

- What if C is an uncertain task too?



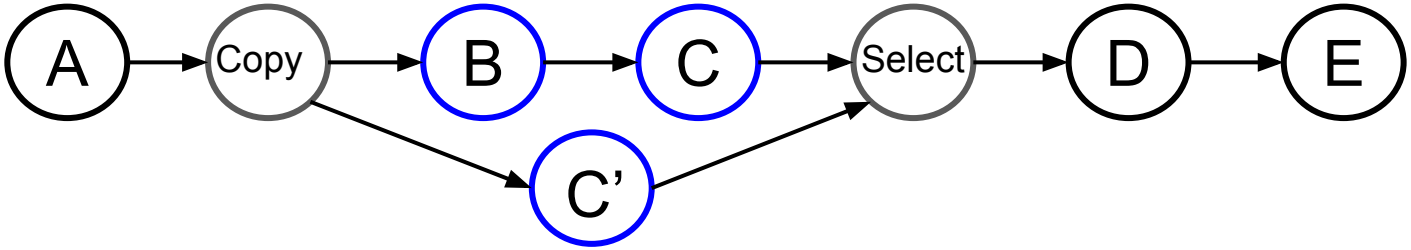
TB Speculative execution example

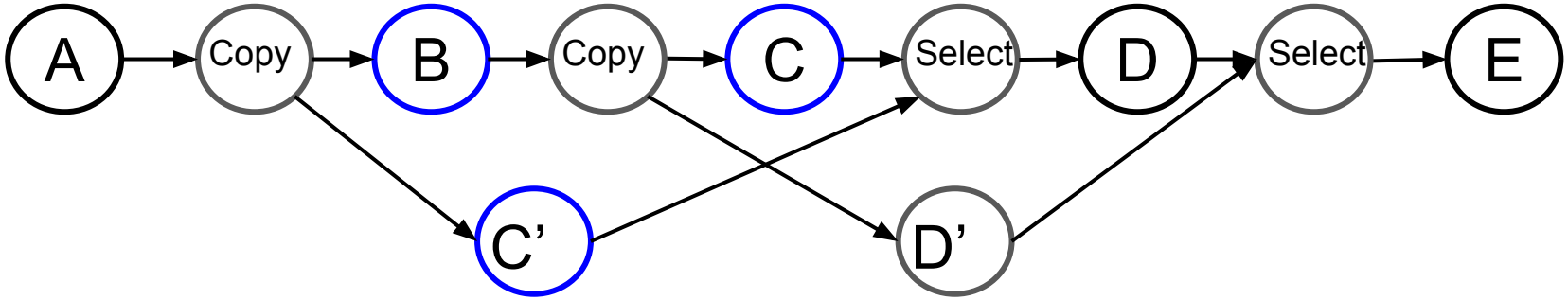




Poor parallelism, D is disabled only if both B and C did not modify the data, must compute B+C+(D or D')
(This is not what we use)

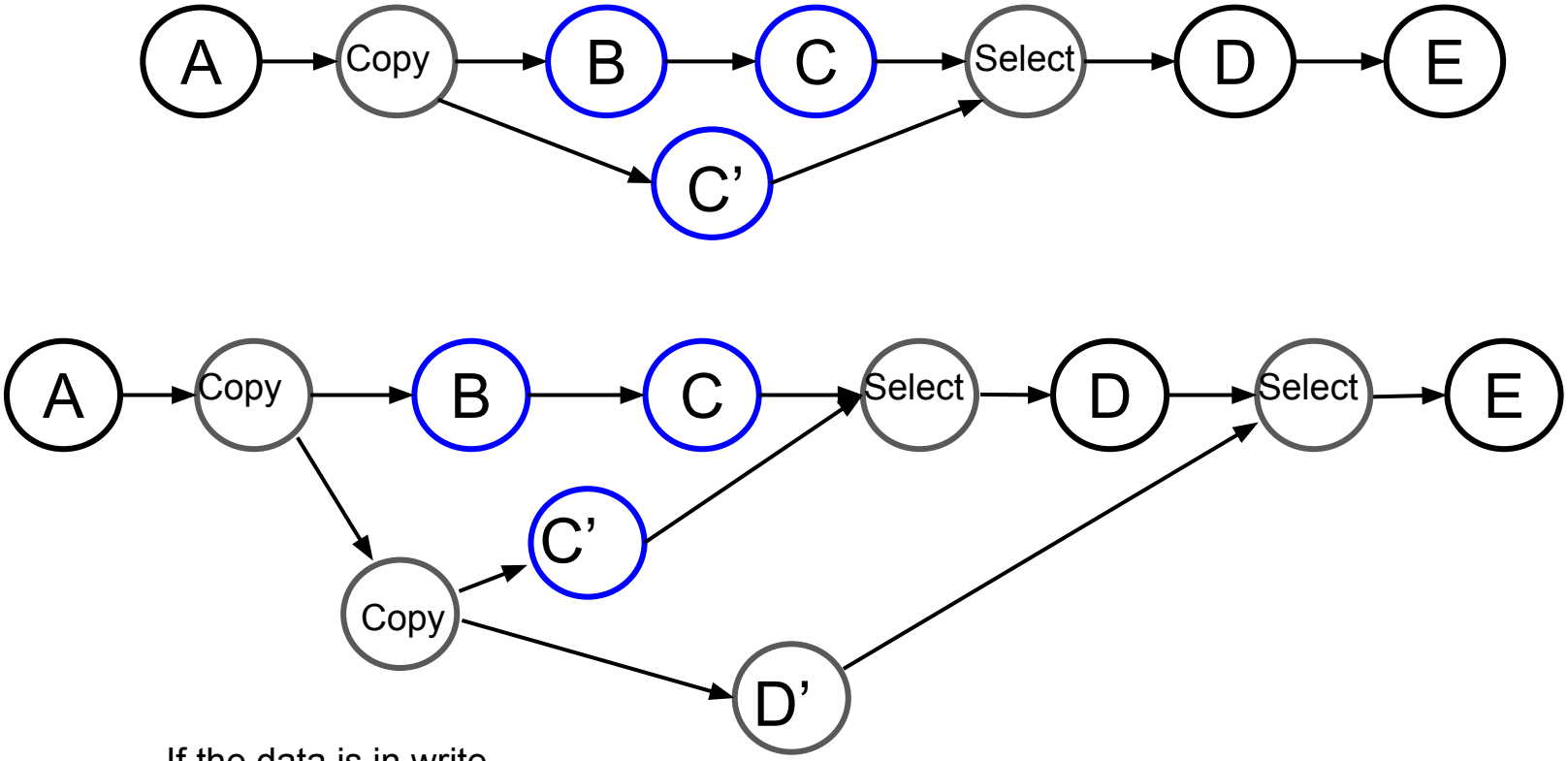
TB Speculative execution example





C',C and D' can be computed together
(This is not what we use)

TB Speculative execution example



If the data is in write
((B,C) or C')D or D' can be computed together
(This is what we use)

SPETABARU: SPeculative TAsk BAsed RUntime

- A light C++ runtime (less than 3.000 lines, originally….)
  - Allow for compile time checking on the types
- Modern C++17
  - Advanced meta-programming
  - Lambda/anonymous functions
- Support data modes:
  - Read, Write, Atomic_write, Commute
- Dynamic “array view”
  - To avoid ugly code when having dynamic dependencies

SPETABARU: SPeculative TAsk BAsed RUntime

- Example

```
// Create the runtime
const int NumThreads = SpUtils::DefaultNumThreads();
SpRuntime runtime(NumThreads);

const int initVal = 1;
int writeVal = 0;
// Create a task with lambda function
runtime.task(SpRead(initVal), SpWrite(writeVal),
        [](const int& initValParam, int& writeValParam){
  writeValParam += initValParam;
});
// Create a task with lambda function (that returns a bool)
auto returnValue = runtime.task(SpRead(initVal),
SpWrite(writeVal),
        [](const int& initValParam, int& writeValParam) -> bool {
  writeValParam += initValParam;
  return true;
});
```

```
// Wait completion of a single task
returnValue.wait();
// Get the value of the task
const bool res = returnValue.getValue();
// Wait until two tasks (or less) remain
runtime.waitRemain(2);
// Wait for all tasks to be done
runtime.waitAllTasks();
// Save trace and .dot
runtime.generateTrace("/tmp/basis-trace.svg");
runtime.generateDot("/tmp/basis-dag.dot");
```

SPETABARU: SPeculative TAsk BAsed RUntime

```
SpRuntime runtime;

runtime.setSpeculationTest([](const int /*inNbReadyTasks*/,
                const SpProbability& /*inProbability*/) -> bool{
        // Always speculate
        return true;
});

int val = 0;

runtime.potentialTask(SpMaybeWrite(val), [](int& /*valParam*/) -> bool {
        return false;
}).setTaskName("C");
```

SPETABARU: Example

Runtime.always_speculate = true

Task A : read(val)

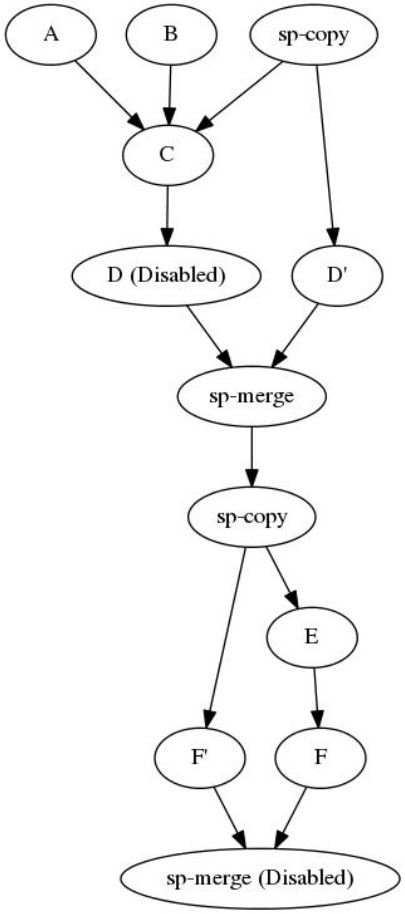
Task B : read(val)

Task C uncertain : maybe-write(val) -> false (no modifications)

Task D : write(val)

Task E uncertain : maybe-write(val) -> true (modifications)

Task F : write(val)

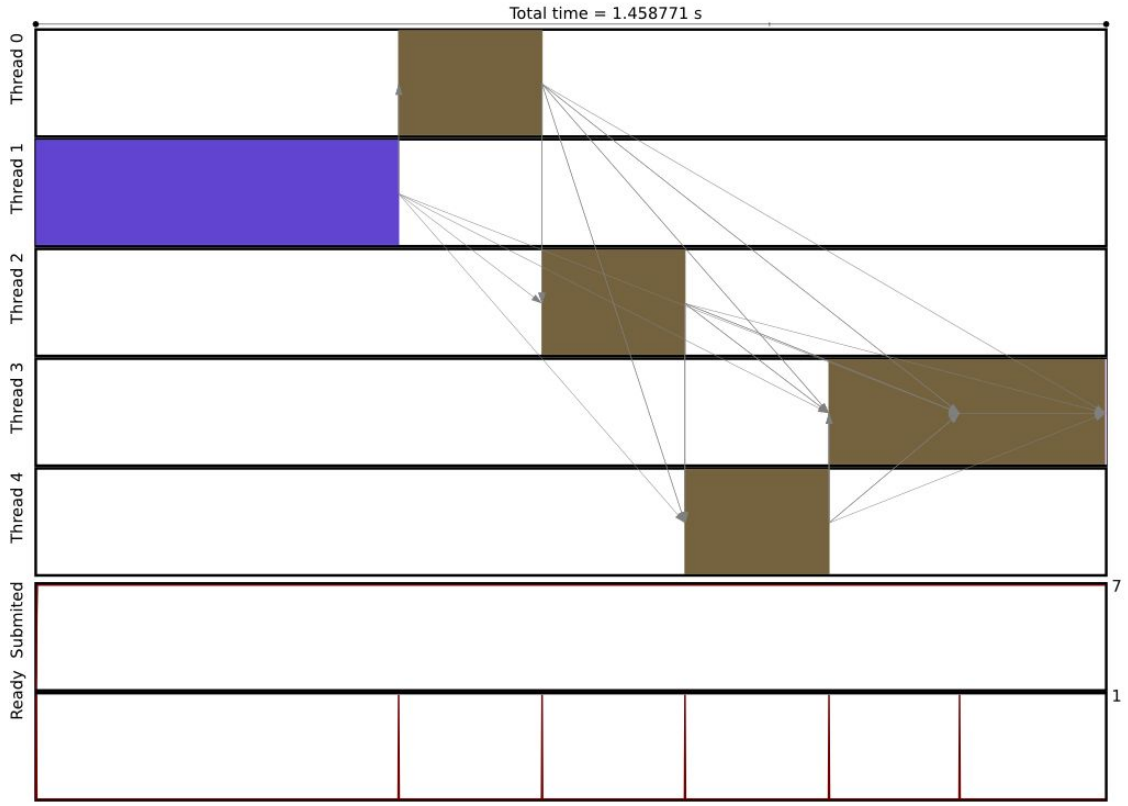


Performance Study - MC

- Configuration
  - 5 domains
  - 2000 particles per domain
  - on a node with 32 cores but we use only 5
  - Reject/accept ratio ~ around 0.4 or 0.5
  - Number of iterations from 1 to 100

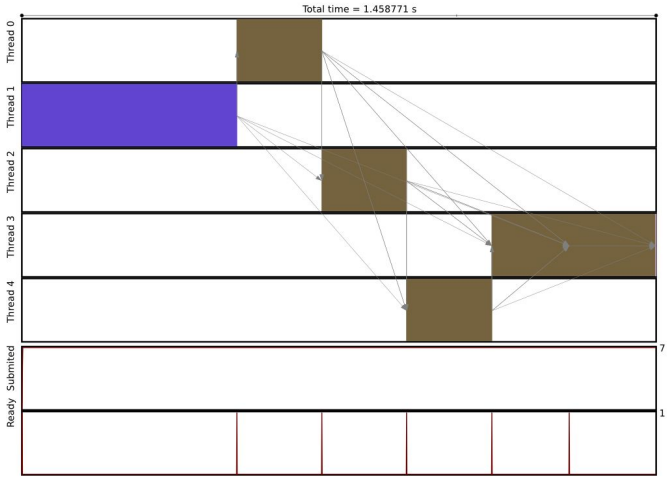
- 3 approaches:
  - Task-based (same as sequential)
  - Speculative (with speculation always enabled)
  - Speculative and force all move to be rejected (speculation performance limits, with speculation always enabled)

Performance Study - MC (1 iteration - 5 threads)



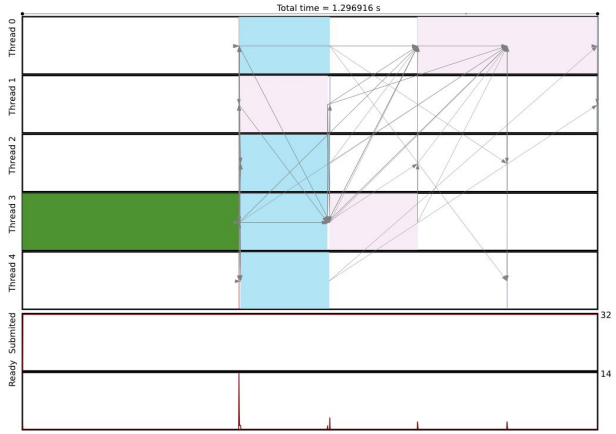
Without speculation

Performance Study - MC (1 iteration - 5 threads)

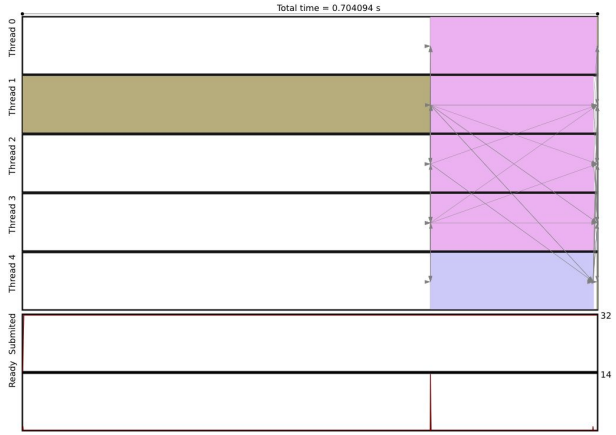


Without speculation - 1.45s

0 : reject
1 : accept
2 : accepted
3 : reject
4 : reject

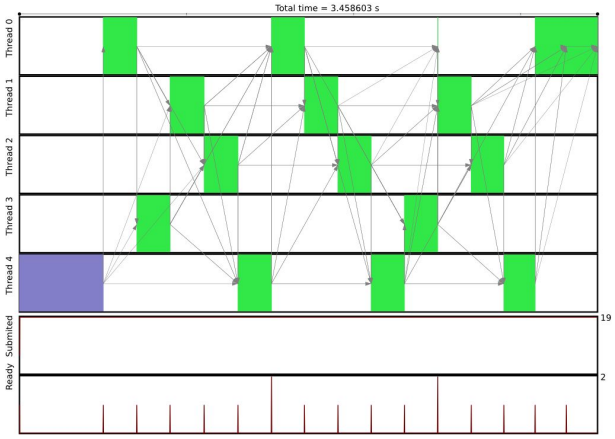


With speculation - 1.29s

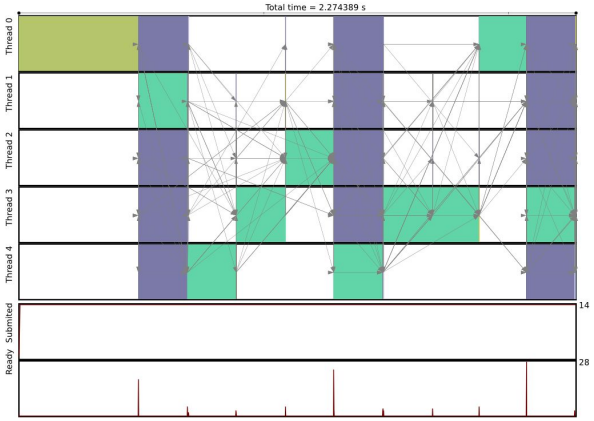


With speculation (all moves rejected) - 0.74s

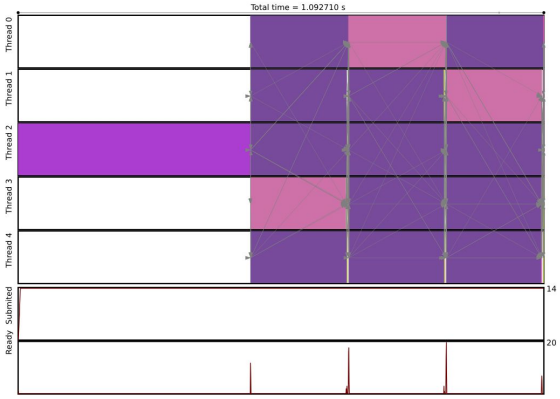
Performance Study - MC (3 iterations - 5 threads)



Without speculation - 3.45s

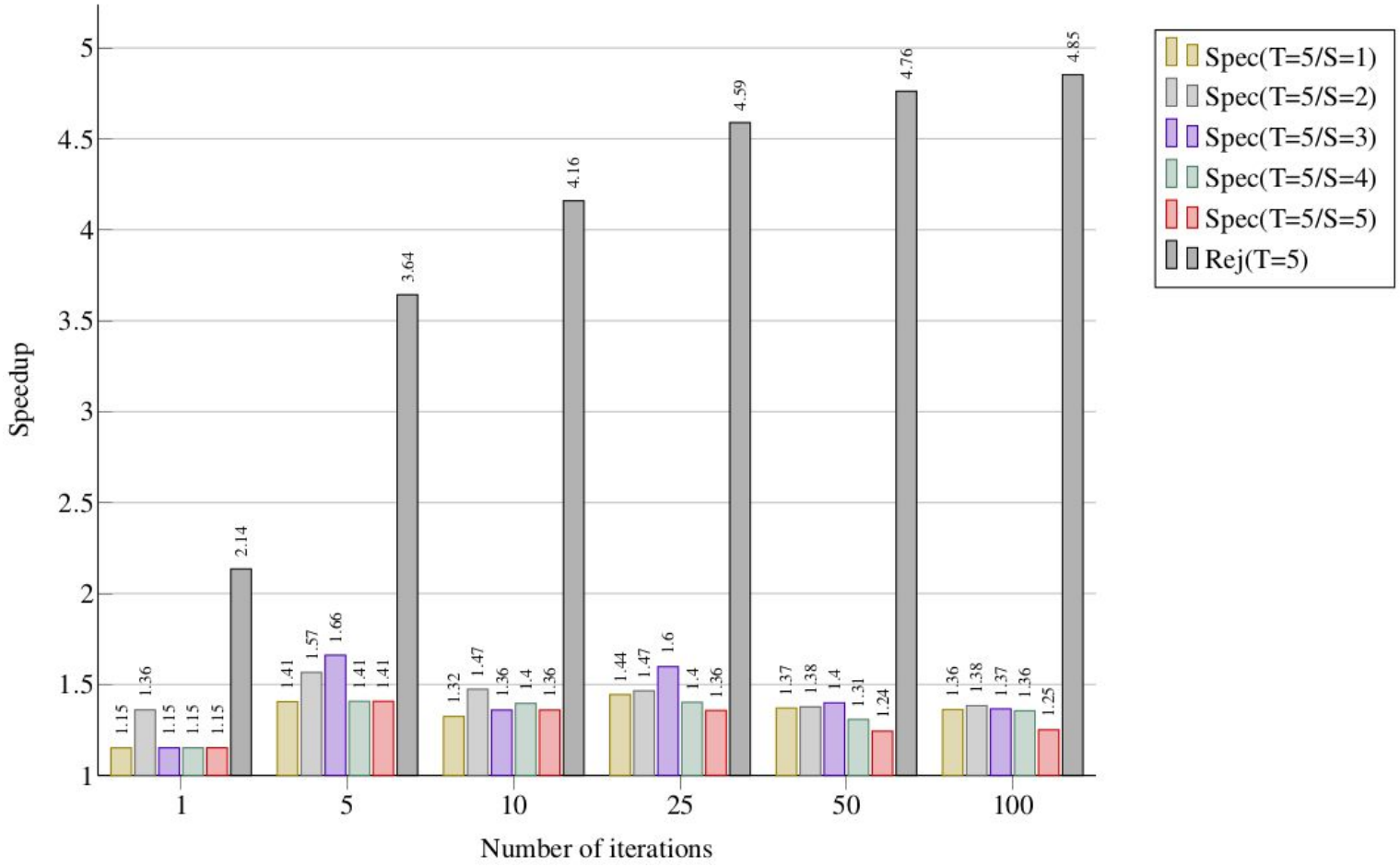


With speculation - 2.27s



With speculation (all moves rejected) - 1.09s

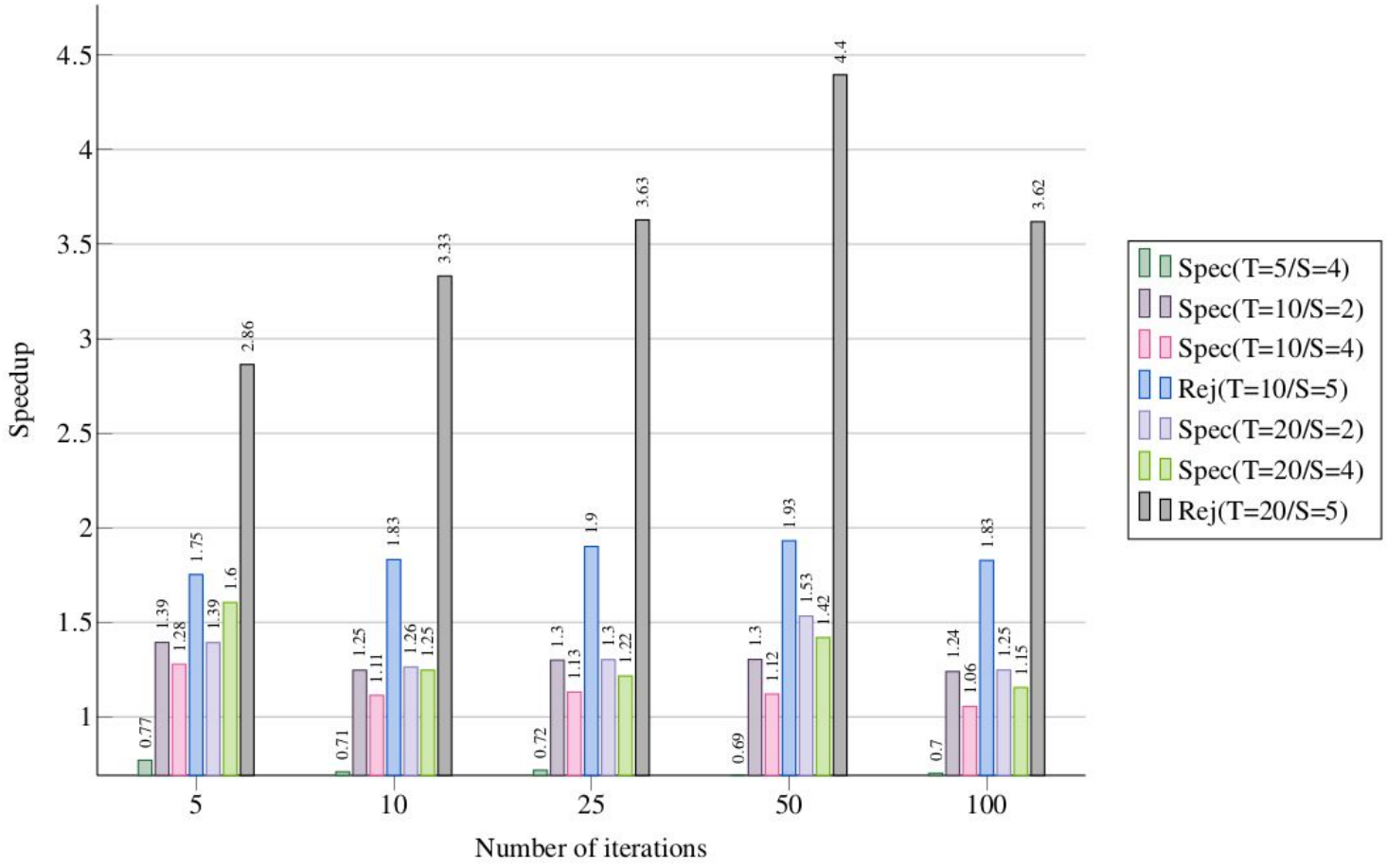
Performance Study - MC - 5 threads



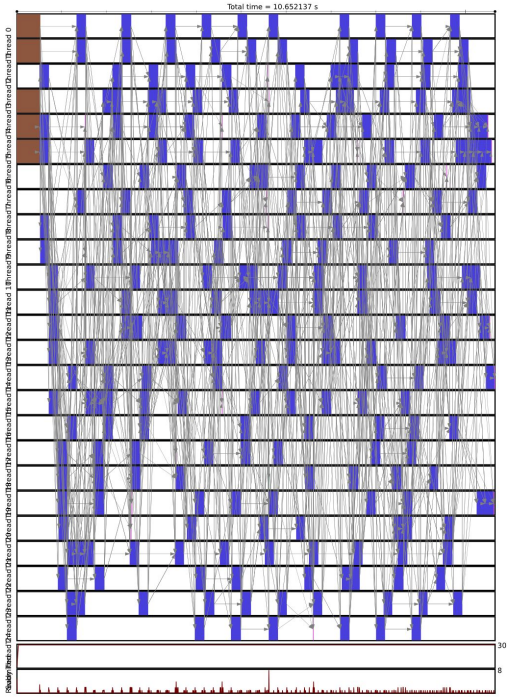
Performance Study - Replica Exchange

- Configuration
  - 5 domains
  - 2000 particles per domain
  - on a node with 32 cores, we use 10 or 20 threads
  - Reject/accept rate ~ between 0.4 or 0.6
  - Number of iterations from 1 to 100
  - + 5 replicas
  - + Replica-exchange every 3 iterations
  - + Replica-exchange rate > 0.7
- 3 approaches:
  - Task-based (same as one thread per replica)
  - Speculative (with speculation always enabled)
  - Speculative and force all move to be rejected (speculation performance limits, with speculation always enabled)

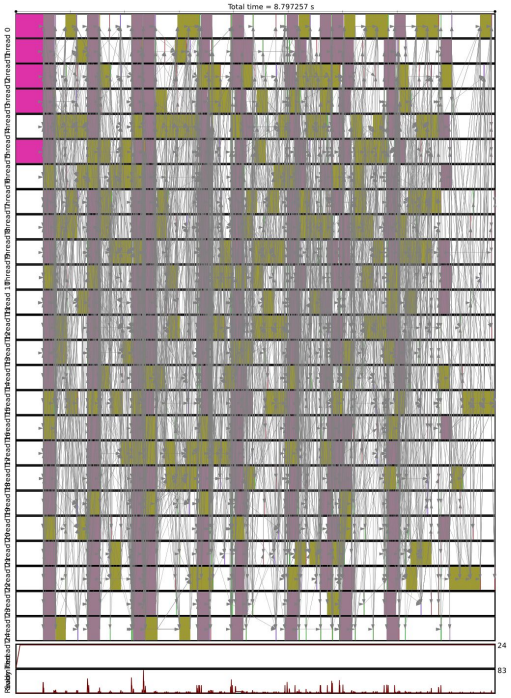
Performance Study - Replica Exchange



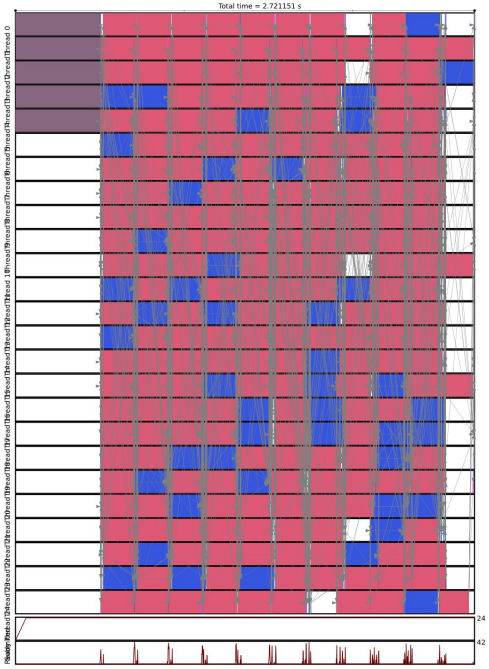
Performance Study - Replica Exchange (10 iterations - 25 threads)



Without speculation - 10.6s



With speculation - 8.7s



With speculation (all moves rejected) - 2.7s

Conclusions

- First results to use speculation in TB method
- General pattern/algorithm
- SPETABARU is able to execute speculative task-flow
- Speedup for both the MC and REMC
  - obviously, the number of failures reduces the benefit
  - can slow down the executions if too much speculation (decision formula) and not enough threads

Perspective

- Limit the number of consecutive uncertain tasks automatically
- Inform RS if modifications have been done for each data individually (easily)
- Prediction of speculation success/failure with a decision formula
  - rand() < probability?
  - only if no tasks ready? seems not efficient
  - Use history with the information of the previous attempts
  - Use a perf. model
- Manage the scheduling of DAG with speculation
- Limit the number of copies (with a maximum memory occupancy value)
- Create other speculative execution models

Questions?

gitlab.inria.fr/bramas/spetabaru